

UNITED STATES PATENT APPLICATION

**METHODS AND SYSTEMS TO ACCESS STORAGE**

**OBJECTS**

**INVENTORS**

**Thomas W. Lanzatella**

of Roseville, Minnesota

**Graham Bromley**

of Pleasanton, California

**John O. Colgrove**

of Mountain View, California

**Ron Karr**

of Mountain View, California

**Blaine T. Cuykendall**

of Mountain View, California

**Oleg Kiselev**

of Mountain View, California

**Craig Harmer**

of Mountain View, California

**Allen Unuecoo**

of Redmond, Washington

**Steve Umbehocker**

of, Redmond, Washington

Schwegman, Lundberg, Woessner, & Kluth, P.A.

1600 TCF Tower

121 South Eighth Street

Minneapolis, Minnesota 55402

ATTORNEY DOCKET 1557.005US1

## Methods and Systems to Access Storage Objects

### Copyright Notice/Permission

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the  
5 facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever. The following notice applies to the software descriptions/examples, and data as described below and in the drawings hereto: Copyright © 2001, VERITAS Software Company., All Rights Reserved.

### **Field of the Invention**

The present invention is related to accessing storage objects, and more particularly to methods and systems provide access to storage objects by traversing and interfacing with a storage management stack associated with the storage objects.

### **Background Information**

Storage networking is the practice of connecting storage devices to computing devices (e.g., clients, servers, and the like) by using Fibre Channel networks instead of traditional point-to-point small computer system interface (SCSI) channels. A network used to connect servers to storage devices is referred to  
20 as a storage area network (SAN). Within a SAN environment, all computing devices have access to the available storage devices. This presents a wide variety of benefits, including server platform fail-over wherein a failed storage device is automatically recovered by another operational server platform without requiring  
25 any recabling of the storage devices themselves. As will be apparent to those skilled in the art, connectivity among the computing devices and the underlying storage devices within the SAN environment is shared.

Prior to the development of SAN technology, local and wide area networks provided connectivity between computing devices that did not include storage

devices. Connections were established with network protocols such as Transmission Communication Protocol (TCP), Unreliable Datagram Protocol (UDP), and others. These protocols ensure that message ordering is preserved and that messages are not lost. Distributed File Systems such as network file system (NFS) and Common Internet file system (CIFS) are layered on top of network protocols. Distributed File Systems organize access to files and correspondingly data storage elements across a network consisting of heterogeneous computing devices. The computing devices are typically organized as clients and servers, in a client-server architecture. Access to files or data storage elements is transparent to any particular computing device, such that access is consistent across the distributed file system without the need to have any private information about the physical locations or details associated with any particular file or data element.

The convenience of distributed file systems comes with an expense, since every byte of data exported by a file server managing a distributed file system must pass through the file server's memory, through the communications stack, and through a network interface controller before it reaches the application. Accordingly, the low performance and low throughput of operation associated with distributed file systems prohibit the implementation of many high performance data-sharing applications such as large scale distributed database applications, backup applications and the like. SAN environments present the opportunity to alleviate this issue by elevating storage devices within the network to peers of the clients and the servers, thereby in theory providing the opportunity for improving throughput of operation.

Yet, SAN technology has not produced advances in throughput of operations as one would anticipate. This is due to the fact that shared access to data among several compute platforms must be mediated by distributed file systems. Consequently, while the speed of connections between platforms has scaled upward with the introduction of SAN, the basic method of using distributed file systems to share data has not changed. Distributed file systems are innately restricted in the level of performance that can be achieved due to the computing overhead introduced

by the communication protocol. Consequently, application writers are motivated to find strategies other than distributed file system in order to share data at speeds that are consistent with SAN technology. These strategies typically employ sharing information about files and volumes with remote application components. Using  
5 this information, an application can know everything about a file without having access to the file through a distributed file system. Additionally, the application can use this information to reference data directly on the SAN-connected storage device.

For these strategies to succeed, applications need to be able to discover sufficient information about files and volumes that a component on another platform  
10 can access the data associated with the file or volume. Customarily, this type of information is not externalized by either file systems or distributed file systems. As used herein this is referred to as private information. Private information differs from one file system operating on one computing device within the SAN and another file system operating on another computing device within the SAN.  
15 Correspondingly, data storage element portability is difficult to achieve within the confines of present SAN technology.

Furthermore, the very purpose of file system and volume manager function within an operating system is to hide the private information related to data storage elements located on one or more data storage devices. Accordingly, operating  
20 system vendors, file system vendors or volume manager vendors do not reveal or provide any useful interfaces that provide access to private information. Moreover, storage environment software stacks are complex and trying to extract the private information from existing stacks is not readily achievable without intervention from the file system software or volume manager software itself.

Moreover private information about data storage elements is in a continual  
25 state of flux in modern data storage architectures, since at any moment in time data storage elements are moved during storage device failure, device reorganization to reduce fragmentation, and the like. Therefore, any ability to acquire private information must also entail notification that private data has been changed.  
30 Otherwise, the acquired private information may be stale or inaccurate.

In addition to performance bottlenecks, many disparate operating systems housing particular file systems can not seamlessly interact with one another since the private information about data storage elements remains largely unobtainable to application software programs. Therefore, when an organization migrates from one operating system, file system or volume manager to a different operating system, file system or volume manager an expensive data storage migration also ensues.

In an effort to address some of these problems some industry associations have been initiated in an attempt to standardize data storage device communication. For example, the Storage Network Industry Association (SNIA) and the National Committee for Information Technology Standards (NCITS) technical committee T11 have been established. Yet, these associations are attempting to gain voluntary standardization for storage device manufacturers to adhere to when developing storage devices, and therefore the work of these associations will not assist with existing storage devices, existing operating systems, existing file systems or existing volume managers. Accordingly, the success of these associations is limited in scope and has yet to be demonstrated even with newly released products and services.

Additionally, even with persistent and stable representations of data storage elements, workable and useable application programming interfaces (APIs) will need to be established, such that different levels of abstraction and interfacing to the storage elements can be achieved seamlessly with user-defined software applications. In this way, user-defined software applications can utilize the APIs to better interact with the storage objects. Moreover, the user-defined software applications will often reside in storage environments different from the storage elements, therefore any provided API must be capable of operating in both storage environments.

Furthermore, each different storage environment can represent a storage element using a variety of different storage configurations. In fact, the underlying operation system and file system associated with a particular storage environment often require the execution of an ordered sequence of operations before access to a storage element is granted within the storage environment. Correspondingly, even

with information about the storage environment of a particular storage element, additional information regarding the storage environment's operating system and file system are needed before proper access to the storage element is obtained.

Moreover, user-defined applications are implemented at different levels of abstraction and thus a user-defined application can make reference to a storage element at different abstraction levels within the storage environment. As a result, any provided API must be capable of determining the storage element's reference level within the storage environment. Therefore, access to the private information or absolute location of the storage element presents a number of challenges when providing APIs to user-defined software applications.

Therefore, what is needed are methods and systems for providing access to data storage elements residing in any storage environment, regardless of any reference level within which access is attempted on the storage element, thereby resulting in access to data storage elements becoming more seamless, transparent, flexible, and reliable across storage environments.

### Summary of the Invention

According to one aspect of the present invention, a method for resolving a storage object's absolute location within a first storage environment to grant access to the storage object is provided, wherein a storage object reference is received. Subsequently, an initial stack level associated with the storage reference is determined and one or more descendent stack levels are queried beginning with the initial stack level. Moreover, the storage reference is translated with each iteration into one or more relative extents until one or more absolute extents are obtained, wherein the one or more absolute extents comprise the storage object's absolute location within the first storage environment.

According to another aspect of the present invention, a method of providing access to a storage object is provided, wherein a request to access the storage object is received from a client module in a first storage environment. Further, one or more mapping plug-in modules are initiated in a second storage environment and the

mapping plug-in modules are used to determine a storage hierarchy of the second storage environment. Next, the storage hierarchy is iterated through using one or more of the mapping plug-in modules until one or more absolute storage locations associated with the storage object are determined.

5           According to still another aspect of the present invention, a storage object access system is provided. The system includes an application programming interface (API) library having one or more client-accessible modules operable to allow a reference to a storage object in a first storage environment. The system further includes one or more plug-in modules operable to interface with the client-  
10       accessible modules in a second storage environment using the reference and granting access to the storage object to one or more of the client-accessible modules in the first storage environment. Moreover, the system has a client module that establishes an interaction with the storage object from the first storage environment using one or more of the client-accessible modules.

15           In yet another aspect of the present invention, a storage object access system is provided. The system includes a storage management stack having a plurality of stack levels, wherein the stack levels include a lowest level identifying one or more storage devices of a first storage environment. Also, the system has one or more plug-in modules wherein each plug-in module interfaces with one of the stack levels  
20       to resolve a reference to a storage object and to pass the resolved reference to a next stack level, unless the resolved reference is an absolute reference to the storage object housed on one or more of the storage devices. Additionally, the system includes a controller that selectively calls a number of the plug-in modules until the absolute reference is obtained.

25

#### Brief Description of the Drawings

Fig. 1 shows one flowchart of a method providing access to a storage object according to the present invention;

Fig. 2 shows one flowchart of a method to resolve an absolute reference to a  
30       storage object according to the present invention;

Fig. 3 shows one flowchart of another method to provide access to a storage object according to the present invention;

Fig. 4 shows one diagram of a storage object access system according to the present invention; and

5 Fig. 5 shows one diagram of a storage access system according to the teachings of the present invention.

### Description of the Preferred Embodiments

10 In the following detailed description of various embodiments of the present invention, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific embodiments in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

15 Some portions of the detailed description which follows are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the ways used by those skilled in the data processing arts too most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and  
20 generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of  
25 common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, terms such  
30 as "processing" or "computing" or "calculating" or "determining" or "displaying" or



the like, refer to the action and processes of a computer system, or similar computing device, that manipulates and transforms data represented as physical (e.g., electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

Furthermore, as used herein a computing device includes one or more processing elements coupled with computer readable memory which can be volatile or nonvolatile memory or any combination thereof. Additionally, the term "object" of "storage object" as used herein includes data storage elements such as, and by way of example only electronic files, portions of data related to a single electronic file, a file system, a database, a storage device partition, and the like.

Initially in some embodiments, a logical representation of a storage object within its native storage environment is obtained with one or more APIs providing efficient access to the storage object provided to interface with the storage object. One such logical representation in one embodiment, is achieved by mapping the structure of the storage object, within the storage environment. This mapping populates a predefined data structure. The data structure is manipulated as appropriate by the APIs which retain knowledge about the format and structure of the predefined data structure. However, any user-defined application or client module using the API need not know the format or structure of the predefined data structure.

Of course as will be readily apparent to those skilled in the art, the mapping need not be a single data structure or any data structure at all since any rules-based application, which normalizes the mapping of the storage object, is acceptable when the API has access to the appropriate rules to interface with the normalized mapping.

For example, in some embodiments an initial reference to a data storage object occurs within the same storage environment wherein the object physically resides or within a different storage environment which is interfaced through a

network connection to the object's physical storage environment, such as a SAN. The reference triggers a pointer to the object to be opened and set initially to null. Next, a map domain request is initiated with the storage object's physical storage environment. The map domain request, in some embodiments, is implemented as a set of executable instructions operable to access low level operating system tables and executable instructions. Further, the map domain request generates a data structure which serves as a map to the structure and makeup of the storage object.

In some embodiments, the data structure is a tree having a root node which identifies the storage object being mapped. The second level of the tree includes an identification of the file system located on the storage object's physical storage environment. A third level of the tree identifies the volume manager associated with routing references to the storage object to an appropriate disk partition or disk drive. Next, the partitions of the disk are identified within the tree. Finally one or more disks are identified as leaf nodes of the tree, these leaves identify the exact physical storage device or devices within which the storage object being referenced physically resides.

Of course as will be readily appreciated by those skilled in the art, a tree need not be used as the data structure representing the map of the data storage object within the data storage object's physical storage environment, since any data structure now known (e.g., lists, linked lists, arrays, hash tables, heaps, binary trees, treaps, arrays, customized classes in objected oriented languages, and the like) or hereafter developed can be deployed without departing from the present invention. Furthermore, as previously presented no data structures are required at all if a rules-based normalization process is used to generate a logical map.

Next in some embodiments, the extents of the object are resolved. An initial extent identifies the name and size of a logical storage object. A number of additional extents identify device's offset and size (length) that, together, describe the physical storage associated with the storage object. Corresponding as one skilled in the art will appreciate, having the generated data structure combined with the extents, an object is readily accessible and referenced from the referencing

storage environment, even when the referencing storage environment includes a first file system which is incompatible with a second file system residing on the data storage object's physical storage environment.

In this way, in some embodiments, any user-defined application or client  
5 module becomes enabled to interface with a data storage object residing on a disparate operating system, if access to the map and information about how to retrieve nodes of the map are known to the user-defined application or client module. For example an application residing within MICROSOFT'S NT operating system can access and interface with a data storage object residing on a storage  
10 device controlled by a UNIX operating system.

Of course it is often desirable and advantageous to mask the details about the map from any user-defined application or client-module, by producing an intermediate set of interfaces referred to as an API. In this way, when the map's underlying representation is altered only a few modules within the API will need to  
15 be adjusted to account for the altered map representation, and the innumerable user-defined applications or client modules do not need modified at all, since these applications or modules do not include the details of the underlying representation of the map. In fact as one skilled in the art will appreciate, when the API is embodied as a dynamic linked library (DLL) or a shared library the user-defined  
20 application or client module will not even be required to recompile the modified API, since the appropriate modified API source code is compiled separate from the user-defined application or client module source code and dynamically linked with the user-defined application or client module during execution.

Furthermore, in some embodiments every storage object residing in a second  
25 storage environment is recursively mapped within the second storage environment as depicted. Accordingly, a complete map (e.g., data structure) of a second storage environment is created. The original referencing storage environment then receives a handle resulting in a direct communication link to each referenced object within the second storage environment by having access to the map (e.g., data structure).

Moreover in some embodiments, each storage object mapped receives a configuration identification which is used each time the referencing storage environment attempts to access the storage object. If the storage object's physical storage location is altered or modified in anyway, this identification notifies the referencing storage environment of the change and an updating map domain request is issued either automatically or manually from any referencing application residing in the referencing storage environment. Accordingly, the storage object's logical representation within the referencing storage environment is validated and/or modified as necessary. This results in a stable and persistent view of the storage object's physical storage environment.

In other embodiments, metadata associated with the data storage object are retrieved and returned to a referencing application in the referencing storage environment. The metadata in some embodiments are returned independent from the data structure, and in other embodiments the metadata are returned with the data structure such that each node within the data structure includes metadata related to the node. In this way, valuable information regarding a disparate file system is obtainable in the referencing storage environment, such as naming conventions, data blocking factor, partitions, and the like. Further, metadata includes any non primary data (e.g., data not physically included within the data storage object's physical location) related to the storage object, such as modification date, security access level, byte size, data type (e.g., text, word processor, image, video, and the like), statistical data (e.g., number of accesses, changes made to the data, and the like) and others.

Also, in some embodiments low-level executable instructions provide the exact unique location of the data storage object's storage device. In some embodiments, this resolution includes a serial number of the device, in other embodiments it is an Internet Protocol (IP) address. This private information provides additional detail to the generated data structure, not normally ascertainable by a file system or operating system.

In still other embodiments, sets of executable instructions permit the virtual mounting of a foreign and remote file system onto a local file system by using the map domain and the metadata to emulate the foreign file system. In this way, and by way of example only, a WINDOW'S NT file system can mount a Unix file system.

As one skilled in the art will readily appreciate, the logical representation of the storage object, acquisition of the metadata, and the exact location of storage devices housing the storage object is best provided to a user-defined application or a client module through a set of modules comprised in an API, since this provides transparent access to the user-defined application or client module, as presented above.

Moreover, production of the logical representation of the storage object entails adequate communication and traversal of the storage object's storage management stack within the storage object's storage environment. In some embodiments, the storage management stack includes a primary group of services provided by an operating system managing the storage object's storage environment. These services manage the relationship between user-defined applications and disk resources, and are further oriented toward managing disk resources and transferring data associated with storage objects between volatile and non volatile memory (RAM, cache, or disk) on behalf of the user-defined applications. No single service within an operating system performs all the necessary data services, and conventionally these services are arranged in a hierarchy called a storage management stack.

Each level of the storage management stack is responsible for specific storage level abstraction. At the lowest level or bottom of the stack, device drivers present a uniform set of interfaces that are responsible for converting Input/Output (I/O) requests into device-specific requests. Typically, above the lowest level of the stack a volume manager level is present, volume managers (VMs) group storage devices into logical groupings referred to as volumes. The manner of grouping disks can vary. Disks may be serially concatenated or arranged in a more complex

configuration to support striping or RAID technology. VMs decompose I/O requests and determine the source and destination of data based on how the volume is configured.

File systems often appear above the VM within a storage management stack.

- 5 File systems build a structure on top of the disk space associated with a volume, the structure is referred to as a file system. The file system allows user-defined applications to claim and use segments of disk storage, generally referred to as files. Applications access data within a file as a series of byte streams, the file systems translate the byte streams to data blocks located on volumes, the VM's translate the data blocks to data blocks on specific storage devices, and finally the device drivers
- 10 convert the data blocks to I/O device specific requests.

- As one skilled in the art will appreciate, each layer of the storage management stack is responsible for hiding some level of complexity from the layer above. Moreover, each element of the stack is responsible for exporting its
- 15 abstraction to the layer above it within the stack. Accordingly, the storage management stack and the elements of the stack include a number of operating system and file system services which permit the abstraction and translations to occur.

- To decompose a storage object within a storage management stack, the
- 20 storage object reference must be translated into one or more absolute extents from the initial reference which is often at different stack abstraction level and is typically associated with one or more relative extents for the storage object. For example, some applications may reference a storage object without the need for a file system, and some file systems do not include a VM. A storage object is definable by using
- 25 extents, wherein each extent identifies a storage management stack level, a beginning location within the identified stack level, and a data length associated with the extent. Furthermore, a single storage object can include several non contiguous extents which when assembled describe all the data associated with a storage object.

As services are used within the storage management stack, relative extents are translated into additional relative extents until at the lowest level, device extents or absolute extents are acquired. The absolute extents provide the exact physical locations for the one or more storage devices which house the storage object. The process of traversing the stack is referred to herein as "mapping through the stack."

By way of example only, consider an application referencing a file (e.g., storage object) identified by the string "fs1/foo" residing on a file system identified as the substring "fs1." An application reference to file foo would identify a top storage management stack reference to the file foo with the following string:

"/fs1/foo." The reference would further include file length information with respect to the file, so that a complete file system relative extent for file foo made by the application would be translated to "/fs1/foo,0,n" where 0 identifies the starting address for foo and n is the length of foo in bytes.

With the extent of foo for the file system resolved, foo's file system processes the file system extent into a VM extent which is handled by the next level of the storage management stack, the VM. The file system, in the present example, translates the file relative extent "/fs1/foo,x,n" to a volume relative extent of (v1,0,n) where v1 is the volume housing foo, 0 is the starting block for foo within the volume, and n is the number of required blocks. Of course it is readily apparent that any number of volume relative extents can be required by a single file reference, and although the present example includes a single volume extent, any number of extents can be derived. Moreover, the actual starting block within the volume can be at any block location, and although the present example has a starting block of 0, this is not a requirement. At this point, the volume relative extent is processed into device extents by the VM such that:

$$(v1,0,n) = (dev1,0,i), (dev2,0,j), (dev3,0,k)$$

where v1 is a simple concatenated set of disks and where i, j, and k are the capacities of devices dev1, dev2, and dev3, respectively. In the present example,  $i+j+k=n$ .

If v1 is a striped volume with a stripe unit size of 2 and a width of 3, mapping through the v1 to produce device extents would produce:

$(v1,0,n) = (dev1,0,2), (dev2,0,2), (dev3,0,2), (dev1,2,2), (dev2,2,2),$   
 $(dev3,2,2), (dev1,4,2), (dev2,4,2), (dev3,4,2) \dots (dev1,((n/3)-2),2), (dev2,((n/3)-$   
 $2),2), (dev3,((n/3)-2),2)$

where the stripe unit size is the number of blocks written to each member of a

5 striped set of disks. However, if v1 is mirrored to two devices, dev1 and dev2, then mapping through the v1 could yield:

$(v1,0,n) = (dev1,0,n)$  and  $(dev2,0,n)$ .

Of course as explained above, there is no requirement that the extents of dev1, dev2, and dev3 begin at block 0 within the devices, and this is presented for purposes of  
 10 illustration only with the present example.

As one skilled in the art will appreciate, some storage management stacks can include partitions requiring a mapping through the partition, and in other embodiments no volume manager is present at all in the storage management stack. Moreover, the translation from one storage stack relative reference to a storage  
 15 object to a next storage stack reference is often achieved with file system or operating system provided operations, not normally used or accessible directly to a user-defined application. However, not all operating systems or file systems make translation operations available, and correspondingly in those situations translation modules are customized to perform the appropriate translation for each non  
 20 providing file system or operating system.

Fig. 1 shows one flowchart of one method 100 providing access to a storage object according to the present invention. Initially, in step 110 a reference to a storage object is received. Receipt of the reference can come from a variety of sources including a user-defined application (e.g., client module), a plug-in service  
 25 module provided at one or more levels within a storage management stack housing the physical location of the storage object, a plug-in module provided to interface a client module and one or more modules of an API library to the storage object, an operating system or file system plug-in module, and the like.

In step 120, the origin within the storage management stack of the reference  
 30 is determined. For example, the reference is determined to be a storage object



reference directed to a specific level within the storage management stack, such as a volume relative extent reference or a device extent reference. Next, a check is made to determine if the reference is an absolute reference in step 122. An absolute reference need not be translated further within the storage management stack and is  
5 used to provide direct access to the storage object. If the reference is an absolute reference (e.g., one or more absolute extents) then the reference is returned in step 130. In some embodiments, the returned absolute reference is passed back up through the storage management stack, and each of the translated references along the way are assembled to form an indirect access to the storage object. The entire  
10 logical representation is then provided for use to an API library or to an initial referencing application for future use when accessing the storage object. In other embodiments, the reference is returned directly to the API library or the initial referencing application as a pointer.

If the reference is not an absolute reference, then the reference is translated  
15 into another reference operable to be processed by the next stack level within the storage management stack in step 124. The translated reference is then passed through the stack to the next stack level in step 126 for processing to again resume in step 120. Processing proceeds until an absolute reference is detected and passed to step 130. As the initially received reference is translated from relative extents to  
20 one or more absolute extents, each translation combines to form a logical map within the storage environment to the storage object, and communication of this map permits applications to access the storage object from disparate storage environments. In some embodiments, the map is formed by a separate distinct processes initiated indirectly and transparently provided through interaction with the  
25 API library.

Fig. 2 shows one flowchart of one method 200 to resolve an absolute reference to a storage object according to the present invention. In step 210 a storage object reference is received. In some embodiments, the reference is received from one or more modules associated with an API library as depicted in step 212.  
30 Furthermore, the API library in some embodiments include one or more plug-in

modules which combine to perform the steps of method 200. The API library includes storage-access modules permitting a client module, residing in a second storage environment that is separate and distinct from a first storage environment housing the storage object, to initiate the storage object reference in step 214.

5 In step 220 access to a storage management stack is acquired, wherein in step 222 an initial stack level associated with the storage reference is determined. As previously presented, once the initial stack level of the received reference is determined, a determination of the file system or VM of the first storage environment in step 224 will identify whether one or more file system or VM  
10 operations are available in step 234 to assist in translating the reference from one or more relative extent references into one or more subsequent relative extent references or one or more absolute extent references. If no file system or VM operations exist, customized operations are provided to facilitate the translation process within the first storage environment.

15 The reference is translated into a one or more relative extents in step 230, however if the reference is initially received as one or more absolute extent references then no translation need occur at all in step 230. A check is made in step 232 to determine if the reference is associated with relative extent references or absolute extent references. If the reference is one or more relative extent references  
20 then the stack level determination and reference translation steps 222 and 230 are iterated, until one or more absolute extent references are acquired for the storage reference.

As previously presented, the absolute extent references combine to identify the storage object's absolute location within the first storage environment. Once the  
25 absolute extent references are acquired for the storage object, the absolute extent references and/or all the translated relative extent references are returned in step 240. The references are returned, in some embodiments, to the proceeding stack level, to an API library, to a client module, and the like. In this way, any storage environment is mapped and access provided by organizing and making available  
30 one or more services to transparently and iteratively translate relative extents into

absolute extents. One skilled in the art will appreciate that in addition to file systems and volume managers, other elements such as snapshot drivers, partition drivers, or disk drivers may participate in the extent resolution process.

Fig. 3 shows one flowchart of another method 300 to provide access to a storage object according to the present invention. A request to access a storage object is received from a client module in a first storage environment in step 310. In some embodiments, receipt of the request from the client module occurs indirectly, such as through one or more API library modules provided in step 315. Also, in step 320 one or more mapping plug-in modules are initiated in a second storage environment, wherein the storage object physically resides in the second storage environment. Again in some embodiments, the mapping plug-in modules are provided in the API library, which also provides modules to the client module. As is apparent, the API library is operable to execute in both the first and second storage environments.

In other embodiments, the mapping plug-in modules are provided by a second operating system or a second file system associated with the second storage environment in step 324. In still other embodiments, the mapping plug-in modules are customized in step 322 to provide the translation capabilities to resolve relative references associated with the storage object and to traverse a storage hierarchy associated with the storage object in the second storage environment.

The mapping plug-in modules are used in step to determine the storage hierarchy of the second storage environment in step 330. The storage hierarchy, in some embodiments, include a file system level, a volume manager level, a partition level, and a device level as depicted in step 332. Furthermore, in step 334, one or more of the mapping plug-ins are used to iterate the hierarchy until one or more absolute locations associated with the storage object are determined in step 350. Finally, in step 360 a link to the storage object is provided to the client module to provide access to the storage object in the second storage environment when the client module resides in the first storage environment.

In some embodiments, the client module makes an additional request for the storage hierarchy in step 340, and the storage hierarchy is provided directly to the client module, thereby providing the linkage to the storage object as depicted in step 342. As one skilled in the art will appreciate, linkage from the absolute locations associated with the storage object in the second storage environment is achieved through a variety of techniques. In some embodiments, these techniques expose the details of the storage hierarchy to the client module and make them directly accessible, while in other embodiments the details of the storage hierarchy are obscured from the client module and embedded within API libraries such that the client module is capable of accessing the storage hierarchy without knowing the details of the hierarchy. In this way, different levels of abstraction are customized and provided to access a storage object with method 300 of Fig. 3.

Fig. 4 shows one diagram of one storage object access system 400 according to the present invention. The system 400 includes an API library 430 having one or more client-accessible modules 432 and one or more plug-in modules 434. The system further includes a client module 412 that establishes an interaction with a storage object 448. The API library 430 includes client-accessible modules 432 residing and executing in a first storage environment 410, and the API library 430 includes plug-in modules 434 residing and executing in a second storage environment 440. The physical locations of the storage object 448 also reside in the second storage environment 440.

The client-accessible modules 432 allow a reference to the storage object 448 which is relative and occurs within the first storage environment 410. In some embodiments, the reference provides the interaction with the storage object 448 directly, such as when the reference is an absolute reference 448 to the storage object 448 in the second storage environment 440.

The plug-in modules 434 interface with the client accessible modules 432, in the second storage environment 440 by using the reference to grant access to one or more of the client-accessible modules 432 to the storage object 448. In some embodiments, the access is provided by the plug-in modules 434 by the construction

of a logical storage map for the storage object 448 in the second storage environment 440.

The map is constructed by the plug-in modules 434 iterating through a storage management stack 442 using the initial reference to determine a current storage management stack level within the stack 442. During iteration through the stack 442, the plug-in modules 434 translate relative references associated with the initial provided reference to the storage object 440 into additional relative references 446 associated with the storage object 448 until one or more absolute references 444 are determined.

Furthermore, in some embodiments the plug-in modules 434 interface with one or more services provided by an operating system of the second storage environment 440. In this way, a file system or VM which provides the ability to translate references into relative references 446 and absolute references 444 is leveraged by the present invention and utilized. While customized plug-in modules 434 are developed, in other embodiments, when the file system or VM of the second storage environment 440 fails to provide services to access and iterate the storage management stack 442.

Fig. 5 shows one diagram of one storage access system 500 according to the teachings of the present invention. The system 500 includes a storage management stack 520 having a top level 520 through a lowest level 524, wherein the lowest level 524 identifies one or more storage devices 527-528 of a first storage environment 510. The system 500 also includes plug-in modules 530 through 532 and a controller 540.

Each plug-in module 530-532 interfaces with one of the stack levels 522-524 in order to resolve a reference to a storage object. In some embodiments, as a reference is resolved it is translated from the present stack 520 level's abstraction to the next lower stack 520 level's abstraction. When a translated reference is detected to be an absolute reference to the storage object housed on one or more of the storage devices 527-528, translation of the references and traversal through the stack 520 terminates normally.

The controller 540 selectively calls the plug-in modules 530-532 as needed and as the reference is translated through the stack 520, until the absolute reference is obtained. Moreover, in some embodiments the controller 540 is a set of executable instructions operable to iterate the stack 540 using the plug-in modules 530-532, while in other embodiments the controller 540 is a set of electro-mechanical devices operable to iterate the stack 540 using the plug-in modules 530-532. In still other embodiments, the controller 540 is provided as or within an API library. In still other embodiments, the controller 540 executes in both the first storage environment 510 and a second storage environment 560. Of course as one skilled in the art will appreciate, the controller 540 need not execute on both environments 510 or 560 since additional modules residing in either environment 510 or 560 could initiate executions of the plug-in modules 530-532 using different interfacing techniques, such as passing data or creating signal or semaphore files within one of the environments 510 or 560, wherein the additional modules detect the data or files and initiate the proper executions. In still more embodiments, the client module 550 interacts with the controller 540 in the second storage environment 560.

Additionally, in some embodiments, the stack 520 is a storage hierarchy representing a storage configuration for the storage object within the first storage environment 510. Further, the stack levels 530-532, in other embodiments, include an application level, a file system level, a volume manager level, and a device level. The device driver level is the lowest level 524, and in some embodiments references or pointers 526 provide a direct linkage to the storage devices 527-528 of the first storage environment 510.

In still more embodiments, one or more replica references are obtained by the controller 540 with the absolute references, wherein the replica references identify replicas associated with the storage object in the first storage environment 510. In this way, mirrored data associated with the storage object occurring on multiple storage devices within the first storage environment, are transparently

interfaced to the client module 550, without the client module 550 being required to manage multiple references.

Although the above embodiments provide illustrations wherein a client module resides in a separate storage environment from the physical location of the storage object, one skilled in the art will appreciate that the present invention is not intended to be so limiting, since in some instances a client module acquires the storage management stack with the present invention and then uses the stack to provide a myriad of different accesses to the storage object with subsequent modules residing in the same storage environment as the storage object or different storage environments.

### Conclusion

The methods and systems discussed above permit reliable and stable access to a storage object. These interfaces permit customized user-defined modules to access low-level private information about the storage object, even when the storage object resides in disparate storage environments, operating systems, and uses disparate file systems. Multiple levels of abstraction in the interfaces are resolved by plug-in modules which directly interface with a storage object's storage management stack to translate relative references to a storage object through the stack until one or more absolute references to the storage object is obtained. The plug-in modules are easily integrated into API libraries and leverage existing operating system or file system provided operations. In contrast to the past, one can now efficiently access and interface between disparate file systems, operating systems, storage environments, or computing environments. As one skilled in the art will appreciate, this is achieved with existing file systems, storage devices, and operating systems with no need for substantial revisions or adherence to any particular data communication standard.

Moreover, as will be apparent to those skilled in the art the present invention can be usefully deployed during the creation of a frozen image associated with the storage environment. In this way, data mirroring and data backup is more easily

achieved with the tenets of the present invention by using the present invention to facilitate applications in creating frozen images of the storage environment.

Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement which  
5 is calculated to achieve the same purpose may be substituted for the specific embodiment shown. This application is intended to cover any adaptations or variations of the present invention. Therefore, it is intended that this invention be limited only by the claims and the equivalents thereof.

10

11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207  
2208  
2209  
2210  
2211  
2212  
2213  
2214  
2215  
22